AU 332 Artificial Intelligence: Principles and Techniques

By: Guowei Deng

Instructor: Yue Gao

November 2, 2018

I. GRIDWORLD

A. Policy iteration

In this part, we are to implement the policy iteration for gridworld. Because the code for value iteration is given, the algorithm for policy iteration is not quite difficult to implement. Here's the result for both algorithms.

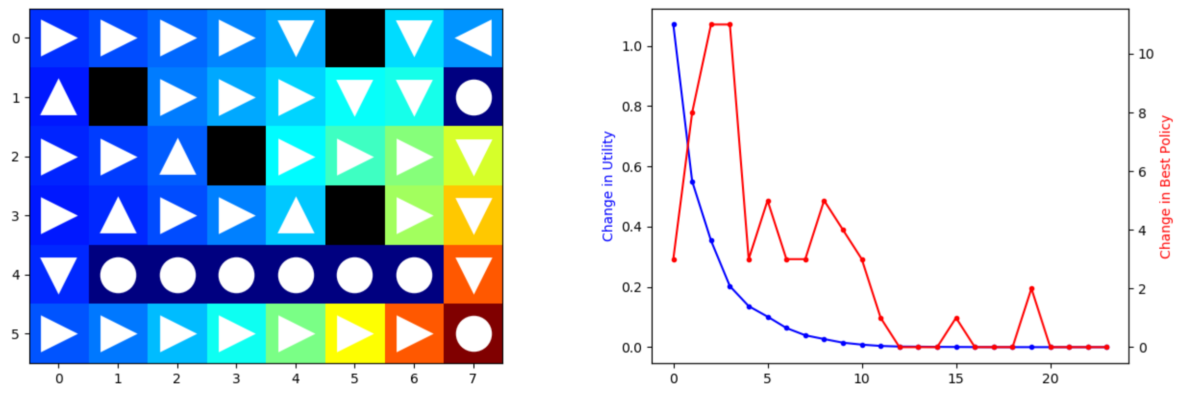


FIG. 1: value iteration

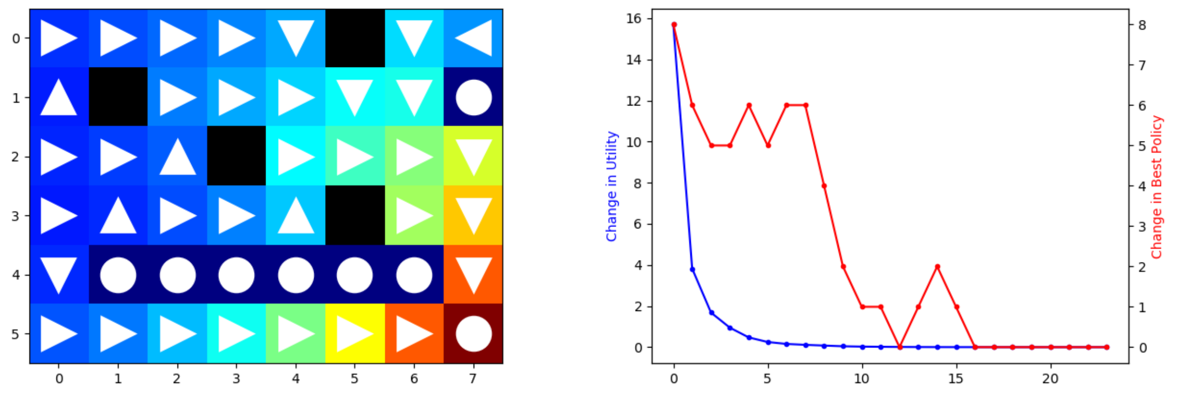


FIG. 2: policy iteration

As we can see, both algorithms can result in optimal policy. But it seems that the policy iteration converges faster than value iteration.

B. Qlearning and Sarsa

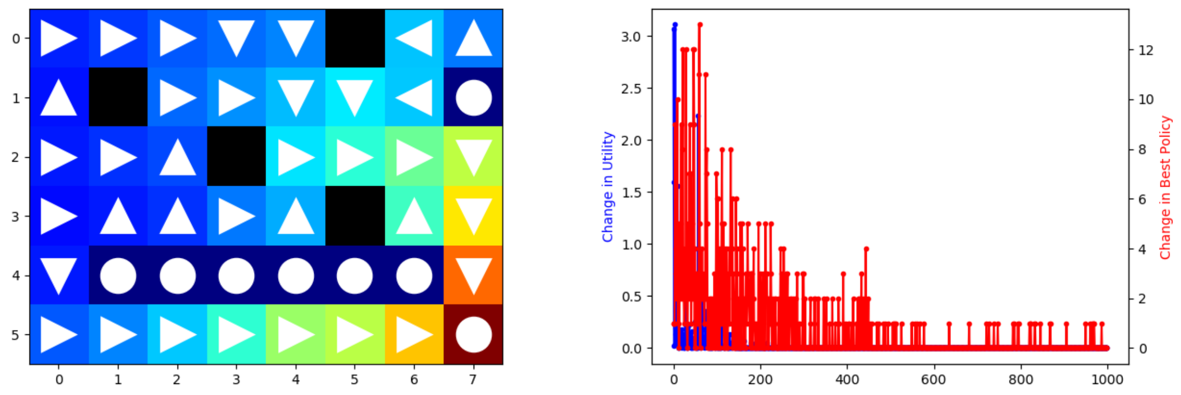


FIG. 3: QLearning

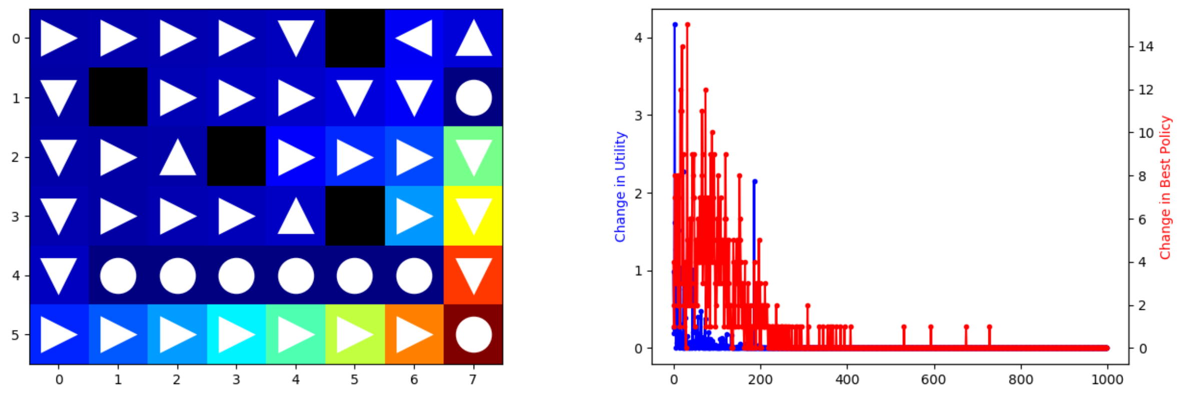


FIG. 4: Sarsa

C. Exploration function

The result for QLearning using an exploration function to explore is as follows:

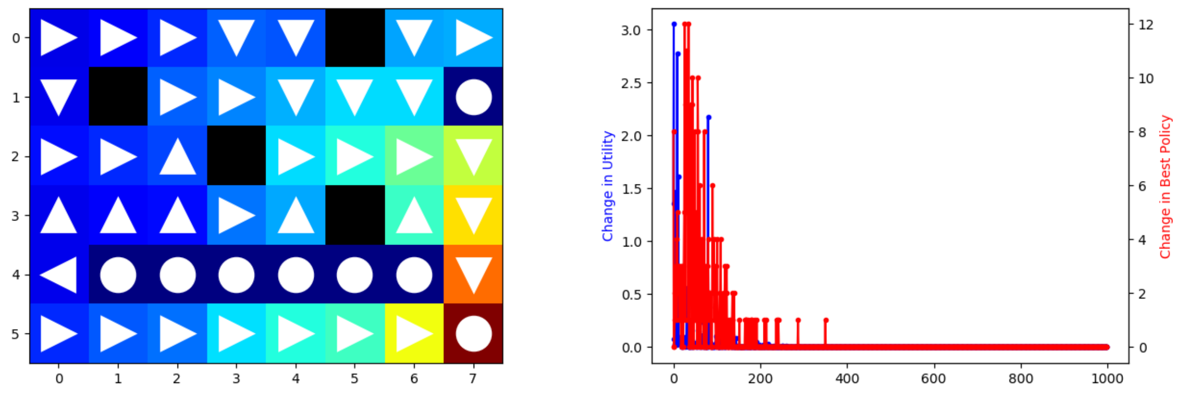


FIG. 5: QLearning-explore

As the graphs show briefly, there are few differences in policy between ϵ-greedy and exploration functon. But using exploration function to explore, it converges much faster. The policy and Q value converge after only 400 iterations, while the other's policy is still changing after 1000 iterations. So the exploration function can reduce many regrets.

D. Hyperparameters tuning

After some thinking and trials, I finally choose hyperparameters $\alpha = 0.8, \gamma = 0.9, \epsilon = 0.4, k = 2$

1. Tuning discount rate γ:
   The discount rate γ indicates how the agent evaluate the reward far from it. As we can see from the policy below, if γ is too high, then the agent will be blindly optimistic and will choose a way which is closer but surrounded by traps . On the other hand, if γ is too low, then the agent will be so pessimistic and will be too scared of traps and choose the adverse direction though it will stay still because of walls.

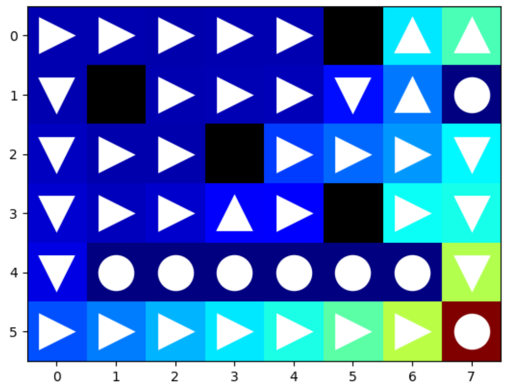


FIG. 6: High discount

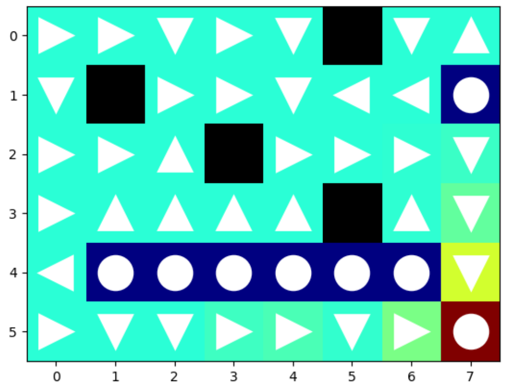


FIG. 7: Low discount

FIG. 8: discount tunning

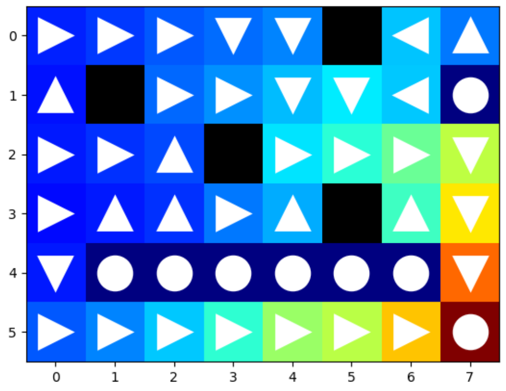


FIG. 9: best discount(0.9)

2. Tuning learning rate α:
   Learning rate is a quite Hyperparameter that influences result of Q-learning. It indicates how much the agent will give up what it has learned and how much it will accept the gain it will get later. And it's quite hard to say which learning rate is better before experiments. So we need to tune it from the result.

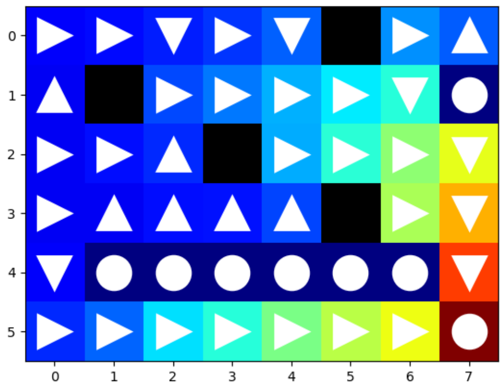


FIG. 10: learningrate=0.6

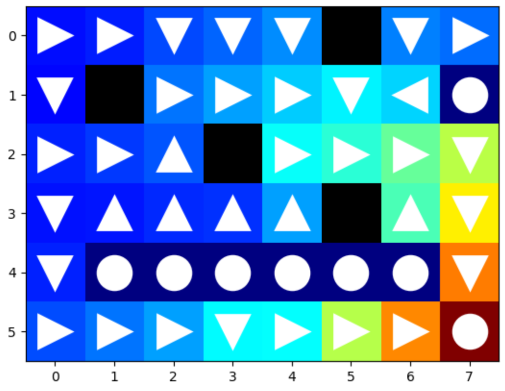


FIG. 11: learningrate=0.7

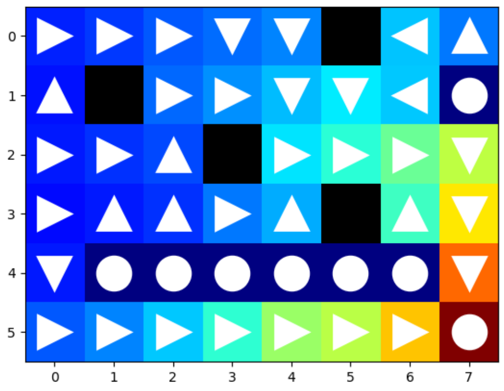


FIG. 12: learningrate=0.8

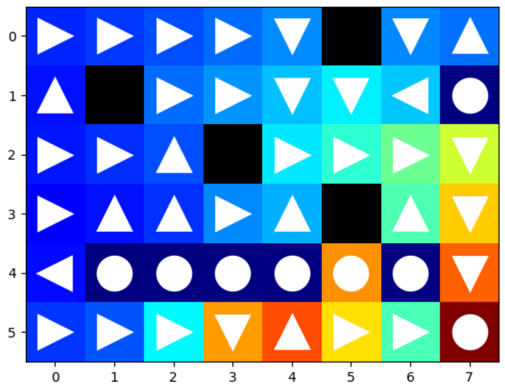


FIG. 13: learningrate=0.9

FIG. 14: discount tunning

It can be seen from the result above that if learning rate is too high, then we get a quite wired result. But if learning rate is too low, some places converge to suboptimal policy. And when α equals to 0.7 or 0.8, the result seems to be fine.

3. Tuning ϵ in ϵ-greedy exploration:
   ϵ-greedy is a method to explore the whole state space. At the very begining, agent chooses action randomly to explore the world. But ϵ should decay as the agent explores the world enough times. So we need to fine a proper ϵ.

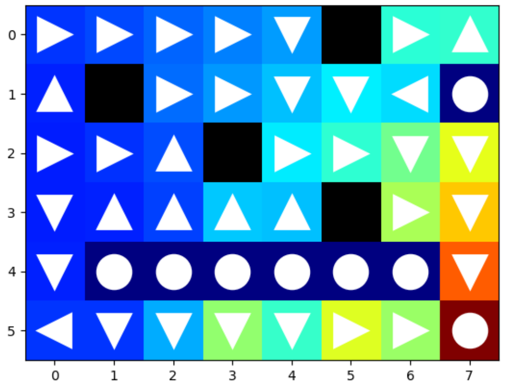


FIG. 15: epsilon=0.3

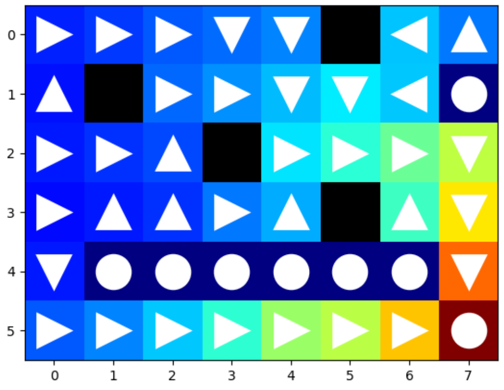


FIG. 16: epsilon=0.4

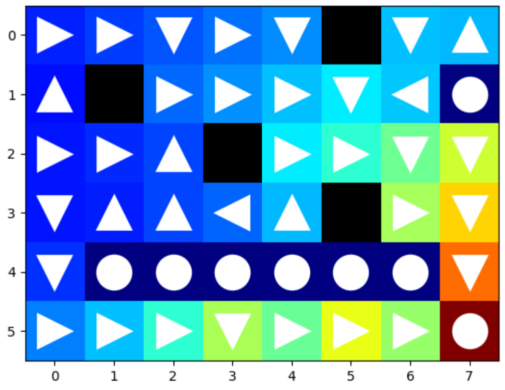


FIG. 17: epsilon=0.5

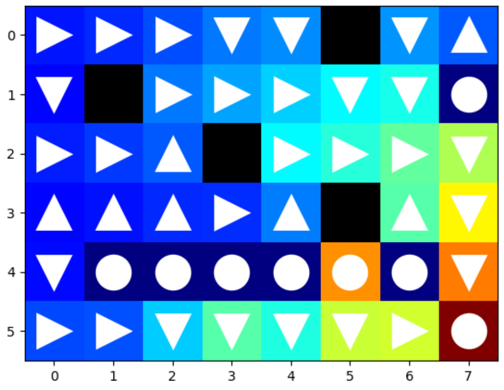


FIG. 18: epsilon=0.6

FIG. 19: Tuning epsilon

E. More improvement

From all the results above, I find a common problem that policy in the upper right is quite bad. However, I think it may not mean that the Q-learning algorithm goes wrong or the hyperparameters aren't tuned well. Because of the given position of initial state and goal state, the agent will hardly explore places on the upper right. To clarify it, just let the agent starts from different position when exploring. And here's the final policy.

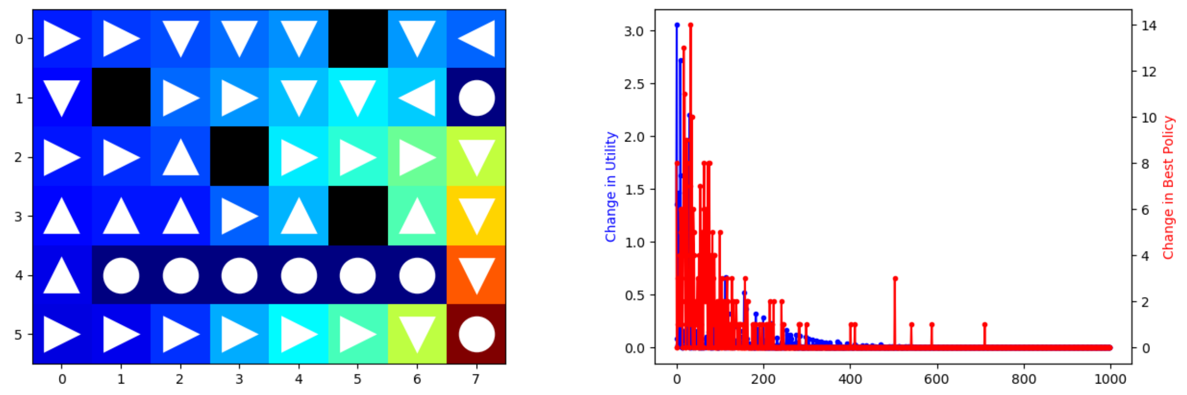


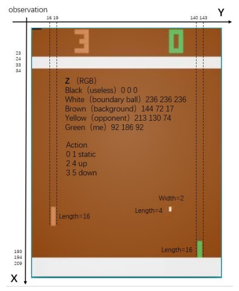
FIG. 20: policy iteration

It's a quite optimal result, and values and policies converge quite fast.

II. GYM PONG-V0

A. The fundamental patten

The main imformation we get from the game is a RGB image. It includes white ball, white walls, brown background, yellow and green board. We are to control the green board to play pong games. We can take 6 different actions to control our board and also change the ball's direction and speed when colliding it. We will lose scores when the opponent misses the ball, the program outputs a positive reward. When we get 21 reward, we will win the game. And vice versa.



B. Q-learning algorithm

In the Q-learning algorithm, we do things as follows:

1. define the generalized state space from the observation
2. define the reward in some specific conditions
3. start with a simple guiding method to help it learn from scarch
4. save the learned Q and test it

1. state space

From the RGB image, we can compute the positions of boards and ball. To make the state space smaller, we need to make it discrete using the vertical and horizontal distance between board and ball. Horizontally, I divide the 1st dimension b of state space into 12 parts. If the ball's Y coordinate is larger than board's, which means the agent misses the ball, y equals to 0. And if ball is on the left of the background(ball's Y coordinate¡60), b equals to 11. And the other 10 states are between the former 2 states. In vertical, which is quite important, I divide into 3 parts. If the ball's X axis is in the range of board, a equals to 0. If the ball's X axis is smaller, a equals to 1, and a equals to 2 for larger similarly. At first, I also divide it into 10 states in vertical. But later I find no matter how long the vertical distance between board and ball, the agent should take the same action since the board's moving speed is stable. So finally, the state space is a $3 * 12$ array and Q is a $3 * 12 * 6$ array consequently.

2. reward

Despite the game will give us reward periodically, but it is not precise enough to help the agent to learn. So I assume the agent will get a positive reward(r=2 indeed) when it catches the ball, and will get a negative reward(r=-5) when it misses the ball.

3. guiding method

If the agent begins to learn from scrach, it will take quite a lot of time to explore and therefore get a high regret. But if we use a simple sample to guide the agent, it will learn much faster. So I just write a simple PID control method and apply it at the first episode.

4. Testing

After training, the learned Q will be saved as a .npy file. To test the performance. And I find it quite hard to get scores against the opponent though luckily the agent will get some scores. And I've trained 8 models and the final model Q_v7 gets the best performance after 50 episodes. I will submit these models as well so you can test it directly without training.

5. Extra ideas

I've got some ideas to improve the performance. But despite the deadline, I have no time to implement them.

1. The Q-learn can be divide into 2 steps. First step is how to move the board to catch the ball and the second is how to hit the ball to make the opponent miss. So we can record the last state and action when hit the ball. If the opponent miss the ball because the agent taked the right action to hit the ball, give that action a high reward in that state.
2. Using several observations nearby to get state, so the agent can get more information such as the ball's direction and speed. But the state space will get much larger and it will take more time to train the agent.